

Sato et al
Filed 7/22/03
Q 76641
10f1

日 本 国 特 許 庁
JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office

出 願 年 月 日

Date of Application:

2002年 7月23日

出 願 番 号

Application Number:

特願2002-213895

[ST.10/C]:

[JP2002-213895]

出 願 人

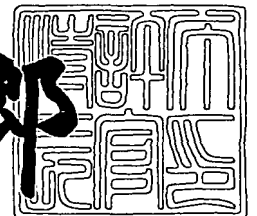
Applicant(s):

NECエレクトロニクス株式会社

2003年 5月13日

特 許 庁 長 官
Commissioner,
Japan Patent Office

太田信一郎



出証番号 出証特2003-3034332

【書類名】 特許願

【整理番号】 74510257

【あて先】 特許庁長官殿

【国際特許分類】 G06F 17/50

【発明者】

 【住所又は居所】 東京都港区芝五丁目 7 番 1 号 日本電気株式会社内

 【氏名】 佐藤 光一

【発明者】

 【住所又は居所】 東京都港区芝五丁目 7 番 1 号 日本電気株式会社内

 【氏名】 澁谷 洋志

【発明者】

 【住所又は居所】 東京都港区芝五丁目 7 番 1 号 日本電気株式会社内

 【氏名】 黒坂 均

【特許出願人】

 【識別番号】 000004237

 【氏名又は名称】 日本電気株式会社

【代理人】

 【識別番号】 100103894

 【弁理士】

 【氏名又は名称】 家入 健

【手数料の表示】

 【予納台帳番号】 106760

 【納付金額】 21,000円

【提出物件の目録】

 【物件名】 明細書 1

 【物件名】 図面 1

 【物件名】 要約書 1

 【包括委任状番号】 0118499

【プルーフの要否】 要

【書類名】 明細書

【発明の名称】 システムレベル設計方法及びシステムレベル設計装置

【特許請求の範囲】

【請求項 1】

コデザイン装置によってシステムのハードウェア部分とソフトウェア部分を分割するとともに、ビヘイビア情報、アーキテクチャ情報、マッピング情報及びアドレス情報を含むデータベースを自動生成するステップと、

前記データベースよりビヘイビア情報、アーキテクチャ情報、マッピング情報及びアドレス情報を含む情報をトップレベル記述生成装置に入力するステップと

前記トップレベル記述生成装置において入力された情報に基づいてクロックベースシミュレーション用の記述を自動生成するステップとを備えたシステムレベル設計方法。

【請求項 2】

前記クロックベースシミュレーション用の記述を自動生成するステップは、前記アドレス情報に基づき、バスとCPUインターフェース間にあるアルゴリズムブロックの選択のためのアドレスデコーダに相当する記述を自動生成するステップを含むことを特徴とする請求項 1 記載のシステムレベル設計方法。

【請求項 3】

前記クロックベースシミュレーション用の記述を自動生成するステップにおいては、仮想バスによりバス接続の記述が行われていることを特徴とする請求項 1 又は 2 記載のシステムレベル設計方法。

【請求項 4】

システムレベルアルゴリズム記述からコデザイン用システムレベルアルゴリズム記述に自動変換するステップを備え、

このステップは、

アルゴリズムブロック間でグローバル変数を介して一方向にのみデータが流れる構成を検出するステップと、

検出された場合に当該グローバル変数をポートに置き換えるステップを有する

ことを特徴とする請求項 1、2 又は 3 記載のシステムレベル設計方法。

【請求項 5】

システムレベルアルゴリズム記述からコデザイン用システムレベルアルゴリズム記述に自動変換するシステムレベル設計方法であって、

アルゴリズムブロック間でグローバル変数を介して一方向にのみデータが流れる構成を検出するステップと、

検出された場合に当該グローバル変数をポートに置き換えるステップを備えたシステムレベル設計方法。

【請求項 6】

システムのハードウェア部分とソフトウェア部分を分割するとともに、ビヘイビア情報、アーキテクチャ情報、マッピング情報及びアドレス情報を含むデータベースを自動生成するコデザイン装置と、

前記データベースよりビヘイビア情報、アーキテクチャ情報、マッピング情報及びアドレス情報を含む情報を入力し、入力した情報に基づいてクロックベースシミュレーション用の記述を自動生成するトップレベル記述生成装置を備えたシステムレベル設計装置。

【請求項 7】

前記トップレベル記述生成装置は、クロックベースシミュレーション用記述を自動生成する場合に、前記アドレス情報に基づき、バスと CPU インターフェース間にあるアルゴリズムブロックの選択のためのアドレスデコーダに相当する記述を自動生成することを特徴とする請求項 6 記載のシステムレベル設計装置。

【請求項 8】

前記トップレベル記述生成装置は、仮想バスによりバス接続の記述を行うことを特徴とする請求項 6 又は 7 記載のシステムレベル設計装置。

【請求項 9】

システムレベルアルゴリズム記述からコデザイン用システムレベルアルゴリズム記述に自動変換する手段をさらに備え、

この手段は、

アルゴリズムブロック間でグローバル変数を介して一方向にのみデータが流れ

る構成を検出する手段と、

検出された場合に当該グローバル変数をポートに置き換える手段とを有することを特徴とする請求項 6、7 又は 8 記載のシステムレベル設計装置。

【請求項 10】

システムレベルアルゴリズム記述からコデザイン用システムレベルアルゴリズム記述に自動変換するシステムレベル設計装置であって、

アルゴリズムブロック間でグローバル変数を介して一方向にのみデータが流れる構成を検出する手段と、

検出された場合に当該グローバル変数をポートに置き換える手段を備えたシステムレベル設計装置。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】

本発明は、大規模回路を設計するためのシステムレベル設計方法及びシステムレベル設計装置に関する。

【0002】

【従来の技術】

近年、ASIC、CPU、メモリ等から構成される複合演算処理システムの開発に対する要求が高まっている。このようなシステムを設計する場合、システム全体の処理動作を記述したアルゴリズムを人手により作成し、まず、このアルゴリズムに対して検証を行う。そして、アルゴリズムをASIC等のハードウェアにより実現する部分と、ソフトウェアにより実現する部分へ分割する。その後、ハードウェア部分についてはRTL HDL記述を作成し、ソフトウェア部分についてはCPUモデルを作成する。さらに、トップレベルHDL記述を作成した上で、これらのHDL記述及びCPUモデルに基づきRTL HDLコシミュレーション装置によって、シミュレーションが行われる。

【0003】

他方、アルゴリズム記述のシミュレーションよりも精細に、かつRTL HDL記述より高速にシミュレーションすることができるシミュレーションモデルを

構築するために、クロックベースシミュレーション技術が提案されている。例えば、特開 2 0 0 1 - 1 0 9 7 8 8 号公報や「SOC の事前検証を実現する C++ シミュレータ」（黒川秀文著、信学技報 VLH98-46）に、このクロックベースシミュレーション技術が開示されている。このクロックベースシミュレーションでは、クロックベース記述に基づき、シミュレーションが実行される。クロックベース記述は、アルゴリズムレベルより抽象度が低く、RTL HDL 記述よりも抽象度が高い。このクロックベース記述においては、バスが抽象化されており、バスへのアクセスが統一化されている。そして、バス・マスタのターゲットに対するアクセスは抽象的なバス・クラスを介して行われる。

【 0 0 0 4 】

【発明が解決しようとする課題】

従来のシステムレベル設計方法では、人手によって設計する部分が多く、労力を要し、かつ短期間で設計を行うという要請に応えることは困難であった。特に、クロックベースシミュレーション用記述の自動生成は実現されていない。

【 0 0 0 5 】

本発明は、このような問題を解決するためになされたものであり、クロックベースシミュレーション用の記述を自動的に生成することができるシステムレベル設計方法及び装置を提供することを目的とする。

さらに、アルゴリズムレベル記述をコデザイン用のアルゴリズムレベル記述に変更する際に、回路規模を縮小することが可能なシステムレベル設計方法及び方法を提供することを目的とする。

【 0 0 0 6 】

【課題を解決するための手段】

本発明にかかるシステムレベル設計方法は、コデザイン装置によってシステムのハードウェア部分とソフトウェア部分を分割するとともに、ビヘイビア情報、アーキテクチャ情報、マッピング情報及びアドレス情報を含むデータベースを自動生成するステップと、前記データベースよりビヘイビア情報、アーキテクチャ情報、マッピング情報及びアドレス情報を含む情報をトップレベル記述生成装置に入力するステップと、前記トップレベル記述生成装置において入力された情報

に基づいてクロックベースシミュレーション用の記述を自動生成するステップとを備えたものである。このような方法により、クロックベースシミュレーション用の記述を自動的に生成することができる。

【0007】

ここで、クロックベースシミュレーション用の記述を自動生成するステップは、前記アドレス情報に基づき、バスとCPUインターフェース間にあるアルゴリズムブロックの選択のためのアドレスデコーダに相当する記述を自動生成するステップを含むことが望ましい。

【0008】

また、クロックベースシミュレーション用の記述を自動生成するステップにおいては、仮想バスによりバス接続の記述が行われていることが好ましい。

【0009】

さらに、システムレベルアルゴリズム記述からコデザイン用システムレベルアルゴリズム記述に自動変換するステップを備え、このステップは、アルゴリズムブロック間でグローバル変数を介して一方向にのみデータが流れる構成を検出するステップと、検出された場合に当該グローバル変数をポートに置き換えるステップを有するとよい。このような方法により回路規模を縮小できる。

【0010】

本発明にかかる他のシステムレベル設計方法は、システムレベルアルゴリズム記述からコデザイン用システムレベルアルゴリズム記述に自動変換するシステムレベル設計方法であって、アルゴリズムブロック間でグローバル変数を介して一方向にのみデータが流れる構成を検出するステップと、検出された場合に当該グローバル変数をポートに置き換えるステップを備えたものである。このような方法により回路規模の縮小と、メモリを介する転送でないため高速処理が期待できる。

【0011】

他方、本発明にかかるシステムレベル設計装置は、システムのハードウェア部分とソフトウェア部分を分割するとともに、ビヘイビア情報、アーキテクチャ情報、マッピング情報及びアドレス情報を含むデータベースを自動生成するコデザ

イン装置と、前記データベースよりビヘイビア情報、アーキテクチャ情報、マッピング情報及びアドレス情報を含む情報を入力し、入力した情報に基づいてクロックベースシミュレーション用の記述を自動生成するトップレベル記述生成装置を備えたものである。このような構成により、クロックベースシミュレーション用の記述を自動的に生成することができる。

【 0 0 1 2 】

ここで、前記トップレベル記述生成装置は、クロックベースシミュレーション用記述を自動生成する場合に、前記アドレス情報に基づき、バスとCPUインターフェース間にあるアルゴリズムブロックの選択のためのアドレスデコーダに相当する記述を自動生成する。

【 0 0 1 3 】

また、トップレベル記述生成装置は、仮想バスによりバス接続の記述を行うとよい。

【 0 0 1 4 】

さらに、システムレベルアルゴリズム記述からコデザイン用システムレベルアルゴリズム記述に自動変換する手段を備え、この手段は、アルゴリズムブロック間でグローバル変数を介して一方向にのみデータが流れる構成を検出する手段と、検出された場合に当該グローバル変数をポートに置き換える手段とを有するようになるとよい。このような構成により回路規模の縮小と、メモリを介する転送でないため高速処理が期待できる。

【 0 0 1 5 】

本発明にかかる他のシステムレベル設計装置は、システムレベルアルゴリズム記述からコデザイン用システムレベルアルゴリズム記述に自動変換するシステムレベル設計装置であって、アルゴリズムブロック間でグローバル変数を介して一方向にのみデータが流れる構成を検出する手段と、検出された場合に当該グローバル変数をポートに置き換える手段を備えたものである。このような構成により回路規模を縮小できる。

【 0 0 1 6 】

【発明の実施の形態】

本発明にかかるシステムレベル設計装置の構成を図 1 に示す。

図 1 に示されるように、当該システムレベル設計装置は、アルゴリズム記述 1 を検証するシステムレベルアルゴリズム検証装置 2 を備えている。このシステムレベルアルゴリズム検証装置 2 は、アルゴリズム記述 1 をシステムレベルアルゴリズム検証ツールにより検証する。一般にアルゴリズム記述は、プログラミング言語である C 言語や C++ 言語によって表現される。システムレベルアルゴリズム検証ツールは、例えば、ケーデンス社の SPW やシノプシス社の COSSAP 等のツールである。

【0017】

アルゴリズム検証装置 2 におけるアルゴリズム記述 1 の例を、図 2 (a) に示す。図 2 (a) において、B 1 乃至 B 7 のそれぞれが、アルゴリズム記述をしたブロックである。また、a、b、c、d がグローバル変数を示している。ブロック B 1 の信号は、ブロック B 2 に入力されている。ブロック B 2 は、グローバル変数 a 及び b を参照するとともに、グローバル変数 b、c 及び d に対して代入している。また、ブロック B 2 は、ブロック B 3 及び B 4 へ信号を出力している。ブロック B 3 は、グローバル変数 b に対して代入し、かつブロック B 5 に信号を出力している。ブロック B 5 は、グローバル変数 c、d を参照するとともに、グローバル変数 a、b に対して代入している。また、ブロック B 5 は、信号をブロック B 6 に出力している。ブロック B 4 は、信号をブロック B 7 に出力している。このように、アルゴリズム検証装置 2 におけるアルゴリズム記述 1 では、一部の信号だけが各アルゴリズムのポートとして表現され、殆どの通信データは、グローバル変数で表現することができる。一部の信号は、大抵ブロックが終了して次のブロックを起動するためのトリガとなる。

【0018】

本発明にかかるシステムレベル設計装置は、さらにアルゴリズム記述 1 を、ハードウェア (HW) とソフトウェア (SW) に分割しやすいようなコデザイン用アルゴリズム記述 4 に変換するコデザイン用アルゴリズム変換装置 3 を備えている。このコデザイン用アルゴリズム変換装置 3 は、上述のシステムレベルアルゴリズム検証装置 2 によってアルゴリズム上問題がないことが確認できた後に処理

を開始する。

【 0 0 1 9 】

コデザイン用アルゴリズム変換装置 3 により生成された、HW と SW に分割しやすいようなコデザイン用アルゴリズム記述 4 は、コデザイン装置 5 に対して入力される。コデザイン装置 5 は、HW / SW 分割装置 7 を有する。この HW / SW 分割装置 7 は、アルゴリズム記述をしたブロックのどれを HW として表現し、どれを SW として表現するか等を検討しながら、アーキテクチャを決定する。即ち、HW / SW 分割装置 7 は、HW と SW を分割する。ここで、HW は例えば A S I C により構成され、SW は C P U 上に構成される。コデザイン装置 5 は、さらに性能シミュレーション装置 8 を備えている。この性能シミュレーション装置 8 は、指定するアーキテクチャによって目的の性能が得られるかを検証するために、性能シミュレーションを実行する。

【 0 0 2 0 】

具体的には、性能シミュレーション装置 8 は、性能シミュレーションのために、まずアーキテクチャダイアグラムを用意する。アーキテクチャダイアグラムは、アーキテクチャインスタンスとして、C P U 、 A S I C 、 R T O S 、 M E M O R Y 、 B U S モデル等を使用してアーキテクチャの全体構成を表現したものである。性能シミュレーション装置 8 は、アーキテクチャインスタンス上で動作させるアルゴリズムを変えながら、遅延を考慮したシミュレーション、即ち性能シミュレーションを実行し、目的の性能としてふさわしいアーキテクチャか判断する。

【 0 0 2 1 】

コデザイン装置 5 で使用するアルゴリズムは、システムレベルアルゴリズム検証装置 2 において用いた記述から不必要なグローバル変数を除去して、関数の引数、即ちアルゴリズムのポートとしたものである。このような処理は、具体的には、コデザイン用アルゴリズム変換装置 3 において実行される。アルゴリズムがアーキテクチャの中で 1 つのブロックで表現されるとき、関数のような表現をとっており、グローバル変数を引数にしておけば、それをポートとして扱う。例えば、図 2 (a) で示されるような記述は、図 2 (b) で示されるような記述に変

更する。このようにポートで表現することによりアルゴリズム間の記憶素子を削減でき、回路規模を小さくすることができる。本発明にかかるシステムレベル設計装置は、このようにグローバル変数を除去する処理に1つの特徴を有する。その処理の具体的な説明については、後に詳述する。

【0022】

コデザイン装置5においてアーキテクチャの決定が終了すると、そのデータは、トップレベル記述生成装置9、HW部生成装置14及びSW部生成装置21に渡る。このデータには、ビヘイビア情報、アーキテクチャ情報、マッピング情報及びアドレス情報が含まれる。

【0023】

トップレベル記述生成装置9は、HWとSWの両方を含むアーキテクチャ全体を表現するトップレベル記述を生成する。トップレベル記述生成装置9は、トップHDL生成装置10とクロックベース記述生成装置12を備えている。

【0024】

トップHDL生成装置10は、コデザイン装置5によって決定されたアーキテクチャ情報等の情報に基づき、後述するRTL HDLコシミュレーション装置26へのトップレベル記述11を生成する。

【0025】

クロックベース記述生成装置12は、コデザイン装置5によって決定されたアーキテクチャ情報等の情報に基づき、後述するクロックベースシミュレーション装置25へのトップレベル記述13を生成する。

【0026】

HW部生成装置14は、インターフェース合成装置15、動作合成装置18を備えている。このインターフェース合成装置15は、コデザイン装置5によって決定されたアーキテクチャ情報に基づき、アーキテクチャのHW部記述生成を行い、HW部のアルゴリズム記述16を生成する。さらに、インターフェース合成装置15は、HW部とBUSをつなぐインターフェース記述を合成し、インターフェース記述17を生成する。動作合成装置18は、HW部のアルゴリズム記述16とインターフェース記述17に基づいて、後述するRTL HDLコシミュレー

ション装置へのRTL HDL記述19を生成する。また、動作合成装置18は、HW部のアルゴリズム記述16とインターフェース記述17に基づいて、後述するクロックベースシミュレーション装置25へのクロックベース記述20を生成する。

【0027】

SW部生成装置21は、ソフトウェア合成装置22を備えている。このソフトウェア合成装置22は、コデザイン装置5によって決定されたビヘイビア情報、アーキテクチャ情報、マッピング情報等の情報に基づき、RTL HDLコシミュレーション装置26において動作できるCPUモデル23を生成する。また、ソフトウェア合成装置22は、コデザイン装置5によって決定されたビヘイビア情報、アーキテクチャ情報、マッピング情報等の情報に基づき、クロックベースシミュレーション装置25において動作できるCPUモデル24も生成する。

【0028】

トップレベル記述生成装置9によって生成されたHDL記述11、HW部生成装置14によって生成されたRTL HDL19及びSW部生成装置21によって生成されたCPUモデル23は、RTL HDLコシミュレーション装置26に入力される。RTL HDLコシミュレーション装置26は、RTL HDL記述レベルにおいて、全体的なシミュレーションを実行する。

【0029】

トップレベル記述生成装置9によって生成されたクロックベース記述13、HW部生成装置14によって生成されたクロックベース記述20及びSW部生成装置21によって生成されたCPUモデル24は、クロックベースシミュレーション装置25に入力される。クロックベースシミュレーション装置25は、クロックベース記述レベルにおいて、全体的なシミュレーションを実行する。ここで、クロックベース記述は、RTL HDL記述より上位レベルであり、アルゴリズム記述より下位レベルの記述である。そのため、クロックベースシミュレーション装置25は、RTL HDLコシミュレーション装置26によって実行されるHDLコシミュレーションよりも高速なシミュレーションを実行することができる。そのアルゴリズムの内容によるが、クロックベース記述のシミュレーション

時間は、RTL HDL記述のシミュレーション時間の概ね500分の1である。

【0030】

クロックベースシミュレーションでは、例えば、次のような項目の見積もり及び検証を行うことができる。

- (1) 各モジュールのクロックベースの動作タイミング検証
- (2) 各モジュールのインターフェースの概略検証 (I/Oレジスタ、I/Oメモリ、I/O端子の構成、名称、ビット幅等)
- (3) 各モジュール、バスの動作クロックの周波数見積もり
- (4) キャッシュアクセスの見積もり (キャッシュヒット率、アクセス率、ライトバック回数等)
- (5) アクセスの見積もり (バスの占有率、バスのトランザクションごとのAddress、Data、Master、Slave、Read/Write、Command、語数、占有時間等)
- (6) Bus、Arbiterのアルゴリズムの検証
- (7) メモリ/I/Fのトラフィックの見積もり
- (8) データ処理のスループットの見積もり (モジュール毎、バス毎、システム全体)
- (9) バッファ、スタックサイズの見積もり
- (10) 画質、音質の見積もり
- (11) アドレスマップ、端子、bit幅等モジュール間接続I/Fの整合性の検証
- (12) 浮動小数点を固定小数点に変換した場合の見積もり
- (13) 組み込みソフトウェアの開発、デバッグ
- (14) 消費電力の概略見積もり

【0031】

また、クロックベース記述のシミュレーションは、RTL HDL記述のシミュレーションに比べて、幾つかのシミュレーションが省略されること等によってそのシミュレーション時間を高速化することができる。例えば、RTL HDL

記述は、クロック信号の変化に応じてレジスタ値は新しいデータ入力値又はそのレジスタが保持していた値に常に更新されるようになるが、クロックベース記述では新しいデータ入力があった変数のみが更新される。また、RTL HDLモデルは、演算器の共有が行われるため、入力となる複数のレジスタから1つのレジスタを選択するマルチプレクサが必要であるが、クロックベース記述では演算器に制限がなく、演算器の入力を選択するマルチプレクサ及び制御入力を作成する回路が不要である。さらに、また、RTL HDL記述は、全レジスタに非同期リセット信号が必要だが、クロックベース記述ではクロック周期単位の動作のみを扱うので、非同期的な動作を行う必要がない。また、RTL HDL記述は、レジスタ、マルチプレクサ、演算器等のサブモジュールを利用した構造を持ち、そのサブモジュールは端子を持ち、その内部には信号線、制御機構等も持っているが、クロックベース記述では、レジスタはプログラミング言語の変数に、マルチプレクサは条件文に、演算器は演算子で表現される。また、RTL HDL記述は、実際のハードウェア通りに束線のビット位置の昇降順、符号のありなし、整数型／ビットベクタ型等の信号型の区別、又は、これらの間の変換を厳密に表現する必要があるが、このような区別は、クロックベース記述では厳密に行われない。さらに、RTL HDL記述内にある各サブモジュール間の動作の並列性は厳密に表現されるが、クロックベース記述では各サブモジュールの厳密な動作タイミングの差に基づいた検証は行われなため、並列性を厳密に表現する必要がない。また、RTL HDL記述は、クロックの変化タイミング以外にも例えばリセット信号が変化した時点の動作を厳密に表現するが、クロックベース記述ではクロック単位の動作のみを扱うので、非同期的な動作に対する処理を簡略化することができる。

【0032】

以上説明したようなシステムレベル設計装置により、システムレベルアルゴリズム記述から、アーキテクチャを決定してクロックベースレベルでの検証まで行う設計フローを実現する。

【0033】

続いて、本発明にかかるシステムレベル設計装置の特徴的な構成及び処理につ

いて詳述する。

【 0 0 3 4 】

最初に、システムレベルアルゴリズム記述からコデザイン用システムレベルアルゴリズム記述への変換処理について説明する。この変換処理は、コデザイン用アルゴリズム変換装置 3 によって実行される。例えば、図 2 (a) に示すアルゴリズムは、図 2 (b) に示すアルゴリズムのようにできるだけグローバル変数を減らすように変換される。

【 0 0 3 5 】

この変換処理においては、第 1 に、図 2 (a) に示す a、c、d のように一方向にしたデータが流れない場合には、単純にポートに変換する。ポートに変換した後の構成を図 2 (b) に示す。図 2 (b) に示されるように、アルゴリズムブロック B 2 とアルゴリズムブロック B 5 間にポートを設定している。ただし、図 2 (a) に示す b については、複数のアルゴリズム B 2、B 3、B 5 よりアクセスされているため、ポートに変換することはできない。

【 0 0 3 6 】

第 2 に、解消できないグローバル変数は、Behavioral memory (ビヘイビアレベルのメモリ) に変換する。図 2 (a) に示す b のようにいたるところからアクセスされる場合、グローバル変数をメモリのような記憶素子で表現する。このような場合、回路はメモリで表現されるのが一般的である。複数のモジュールから一つの変数を書き込んでいき、あるタイミングで複数のモジュールから変数値の読み込みができる。

【 0 0 3 7 】

グローバル変数を減らす処理を行わずにコデザイン装置 5 の HW / SW 分割装置 7 によって HW 部と SW 部の分割処理を行った後のアーキテクチャを図 3 に示す。この構成は、図 2 (a) に相当するものであり、グローバル変数 a、b、c、d がそれぞれメモリ 3 9、3 3、4 0、4 1 として表されている。図 3 においては、HW 部 3 0 と SW 部 3 1 にアルゴリズムがマッピングされている。HW 部 3 0 では、書き込み手段 3 4 及び読み込み手段 3 5 を介して、バス 3 2 と B 2 インターフェース 3 6、B 4 インターフェース 3 7 及び B 5 インターフェース 3 8 が接

続されている。B 2 インターフェース 3 6、B 4 インタフェース 3 7 及び B 5 インタフェース 3 8 は、インターフェース合成により自動生成され、それぞれ、アルゴリズムブロック B 2、B 4、B 5 と接続されている。SW 部 3 1 には、アルゴリズムブロック B 1、B 3、B 6、B 7 がマッピングされている。また、バス 3 2 には、直接、グローバル変数 b に相当する外部メモリが接続されている。

【 0 0 3 8 】

図 4 にグローバル変数を減らす処理を行った場合の構成を示す。図 4 に示す通り、グローバル変数 a、c、d に相当するメモリは削除され、ポートに変換されている。そのため、図 3 に示すグローバル変数を減らす処理を行う前に比べて、メモリ a、c、d 分だけ回路規模を小さくすることができる。

【 0 0 3 9 】

次に、グローバル変数削除処理のアルゴリズムの一例を説明する。図 6 に、記述例を示す。図 6 (a) は、図 2 (a) に示すアルゴリズムブロック B 2、B 3、B 5 を表現する記述である。まず、図 6 (a) の記述を解析し、グローバル変数の read を使用しているか、write を使用しているかを調査する。調査結果が図 6 (b) である。次に、図 6 (b) における read、write の数の総計を計算する。グローバル変数の read、write の数がそれぞれ 1 以下の場合は、ポートに変換し、グローバル変数は削除できる。この例では、a、c、d がその変数に相当し、削除することができる。削除できずに残ったグローバル変数 b は、そのままの状態が維持される。最終的に、関数は、図 6 (c) に示す記述のように変換される。

【 0 0 4 0 】

続いて、コデザイン装置 5 からクロックベースシミュレーション装置 2 5 へ接続するためのクロックベース記述生成装置 1 2 の処理について詳述する。

【 0 0 4 1 】

まず、コデザイン装置 5 では、性能シミュレーションによる性能見積りを行いながらアーキテクチャ決定を行う。ここでは、Cadence 社製の VCC (Virtual Component Codesign) の機能を前提に記述する。次にこのアルゴリズムを動作させるアーキテクチャモデル (アーキテクチャダイアグラム) を用意する。アーキ

テクチャはコデザイン装置上でキャラクタライズされた、プロセッサやバス等の様々なアーキテクチャコンポーネントを配置、結線して表現する。

【 0 0 4 2 】

次にマッピングを行う。アルゴリズムの各ブロックをアーキテクチャのどのコンポーネントで実行するか、アルゴリズムのデータの流れがアーキテクチャのどのインターフェースを流れるのかを G U I 等を通して、線で結んでマッピングする。

【 0 0 4 3 】

通信経路の決定にはパタンを使用する方法もある。パタンは、アルゴリズムブロック間のデータ転送に関するプロトコルを表す。H W → S W 間の通信であれば、C P U I / F に相当するメモリマップ I O のアドレス情報なども有する。

【 0 0 4 4 】

マッピングした後、個々のアーキテクチャコンポーネントに定義された遅延値がアルゴリズムにアノテートされ、シミュレーションを行うことにより、そのアルゴリズムがそのアーキテクチャにマッピングされたときの性能を検証できる。

【 0 0 4 5 】

クロックベース記述生成装置 1 2 は、このようにして生成されたコデザイン装置 5 の情報を用いて、トップレベルクロックベース記述 1 3 を作成する。

【 0 0 4 6 】

次に、クロックベース記述生成装置 1 2 によりトップレベルクロックベース記述 1 3 の作成につき説明する。トップレベルクロックベース記述 1 3 は、次のようなフォーマットを有する。

インスタンス宣言 ;
Initialization 処理 ;
Reset 処理 ;
Step 処理 ;
busread 処理 ;
buswrite 処理 ;
ブロック間の直接接続宣言 ;

【0047】

このようなフォーマットの記述をクロックベース生成装置12によって自動生成している。以下にそれぞれのフォーマットの生成につき説明する。

【0048】

最初にインスタンス宣言の生成について説明する。トップレベルクロック記述におけるインスタンス宣言は、コデザイン装置5により生成されたマッピング情報から、アーキテクチャインスタンス、アルゴリズムインスタンス（ビヘイビアインスタンス）を取得することにより生成する。ここで出力するインスタンス宣言の対象は、プロセッサ、バスなどの既存アーキテクチャインスタンス、ASICにマッピングされたアルゴリズム、ASICにマッピングされたアルゴリズムのCPUインターフェースである。コデザイン装置5におけるマッピングが図5に示すような構成を有する場合において、クロックベース生成装置12は、図4に示すような構成のトップレベル記述を生成する。

【0049】

ここで、アーキテクチャダイアグラム中に存在するインスタンスから作成されるインスタンス宣言例を以下に示す。

【0050】

バスの場合には、次のようなインスタンス宣言となる。

```
BUS Bus_1("Bus1", arbitor_1, busread_1, buswrite_1);
int arbitor_1(void);
RDATA busread_1(int add, int byte_count);
int buswrite_1(int add, int data, int byte_count);
```

バスのインスタンス自体はBUS bus_1のように宣言し、arbitorの宣言も同時に出力する。arbitor_1の内容については適当なものを自動生成し、ユーザが変更したい場合、あとで本トップレベル記述を編集する。同時にbusread_1、buswrite_1といった関数宣言をする。これは図8、図9に示す記述のうち、busread_1、buswrite_1と記述したブロックに相当し、バスとASIC中のアルゴリズムのCPU I/Fを結ぶ関数である。具体的には、メモリマップ方式におけるビヘイビア選択のためのアドレスがここに設定される。

【0051】

プロセッサの場合には、次のようなインスタンス宣言となる。

```
CPU cpu_1("cpu_1",&cpu_1_busif,0,0x10000,0x00ffe000,0x1000,0x8000);
BusIntf cpu_1_busif("cpu_1 Bus I/F",&bus_1,1)
```

このようにしてCPUコアの設定及びCPU用バスの設定を行う。数字は、ROM、RAMのスタートアドレスに相当するが、本データに関してはコデザイン装置で扱っていない場合、未設定で出力し、ユーザが後に設定する。

【0052】

外部ROM、RAMの場合には、次のようなインスタンス宣言となる。

```
SDRAM ExtSDRam1("External SDRam 1", EXTSDRAM1_START_ADD, EXTSDRAM1_SIZE)
```

コデザイン装置5におけるメモリのアーキテクチャインスタンスがこの宣言に変換される。

【0053】

ASICにマッピングされたアルゴリズム、CPU I/Fは以下のような宣言で作成する。

```
c_cpuifB2 cpuifB2(); c_cpuifB4 cpuifB4(); c_cpuifB5 cpuifB5();
c_B2 funcB2(); c_B4 funcB4(); c_B5 funcB5();
```

ここで、c_cpuif*** (***)には任意の関数名が入る) はそれぞれのcpuifを表現するclassである。また、c_B** (**には任意のアルゴリズムの番号が入る) はそれぞれのalgorithmを表現するclassである。これらのclassや実体はHW部生成装置14で作成されるクロックベース記述である。

【0054】

次にInitialization処理、Reset処理及びStep処理の生成について説明する。

アルゴリズム記述は動作合成によってクロックベース記述に合成される。クロックベース記述は、init()、reset()、OneStep()等の構成要素をもつ。init()は、対応するアルゴリズムブロックを最初に初期化するとき使用する。reset()は、対応するアルゴリズムブロックがリセットされた場合の内部変数の初期設定を行う。OneStep()は、アルゴリズムブロックに対してクロックが入った場合の

1 クロック分の処理を記述する。このOneStep()は、アルゴリズム記述の中核となる関数である。以上のようなブロックを並列に動作させる場合、以下のように使用する。

```
{
funcB2.OneStep(); cpuifB2.OneStep();
funcB4.OneStep(); cpuifB4.OneStep();
funcB5.OneStep(); cpuifB5.OneStep();
}
```

この記述は、B 2、B 4、B 5 のアルゴリズムブロックが並列動作することを意味する。初期化、リセットも同様に次のような記述がされる。

```
systemInit()
{
funcB2.Init(); cpuifB2. Init();
funcB4.Init(); cpuifB4. Init();
funcB5.Init(); cpuifB5. Init();
}
systemReset()
{
funcB2.Reset(); cpuifB2. Reset();
funcB4.Reset(); cpuifB4. Reset();
funcB5.Reset(); cpuifB5. Reset();
}
```

【 0 0 5 5 】

次にbusread処理及びbuswrite処理の生成について説明する。busread及びbuswriteは、コデザイン装置5によって生成されるマッピング情報よりアドレス情報を取得することによって作成される。busread及びbuswriteは、バスとCPU I/Fの間にあるアルゴリズムデコーダに相当する。cpuifとbusread、buswriteとの関係は図8及び図9のように表すことができる。図中にあるCPU I/Fはインターフェース合成及び動作合成により作成される。インターフェース合成

により入力記述は、特開 2 0 0 1 - 1 1 7 8 5 5 号公報に示される手法により使用するものを前提としている。また、アルゴリズム部はアルゴリズム記述から動作合成によって作成されるクロックベース記述である。このように、本発明にかかるシステムレベル設計方法では、コデザイン装置 5 によって生成されるマッピング情報よりアドレス情報を取得することによって busread 及び buswrite を作成している点に特徴を有する。HDL 記述においては、バス構造、バスの使用方法に応じたピンを指定しなければならなかったため、この処理を自動化することは困難であった。これに対して、クロックベース記述は、仮想バスに指定すればよく、ピン情報を詳細に設定する必要がないため、アドレス情報に基づき busread 及び buswrite を自動的に作成することができる。また、通常は、バスと CPU I/F の間を直接的に関連付けているが、本発明にかかるシステムレベル設計方法では、busread 及び buswrite というアドレスデコーダに相当する記述を設けており、この点にも特徴を有する。

【 0 0 5 6 】

続いて、ブロック間の直接接続宣言の生成について説明する。

図 1 0 に示すアルゴリズムブロック B 2 と B 5 の間の直接接続記述は以下のよう

```
connection()
{
  B5.a = B2.a;
  B5.b = B2.b;
  B5.c = B2.c;
}
```

ここでは説明を省くが、CPU I/F とアルゴリズムブロック間の直結にも直接接続は使用される。

【 0 0 5 7 】

ここで、トップレベル HDL 記述とトップレベルクロックベース記述の違いについて詳述する。

基本的に両者が大きく異なる点は、バス接続の記述方法である。クロックベ

ス記述の場合、cpuifのReadIOReg(), WriteIOReg()でCPU I/Fとのデータのやり取りを行い、RDATAという構造体でbusにデータを渡す。

HDL記述の場合、図11に示すようにバスと物理的なピンで接続するモデルとなっている。バスが変更されるとピン情報が変わり、接続の仕方もその都度変えなければならない。また、詳細なプロトコルを表現することも可能である。

クロックベース記述の場合、RDATAという構造体でデータを渡すため、図10に示すように、バスの物理的な構造によらずにデータのやり取りを表現する。細かいバスプロトコルを考慮できないため、バスの使用の仕方によっては、RTL

HDL程の正確な遅延見積りがこの部分で行えないかもしれない。ユーザはその点を考慮にいれて使用する必要がある。

【0058】

このように本発明にかかるシステムレベル設計方法では、アルゴリズム記述中における冗長なグローバル変数表現をポート表現に変換することで、回路規模の縮小を図ることができる。また、コデザイン装置における記述からクロックベースシミュレーション装置用のクロックベース記述に自動変換している。特に本発明では、従来自動合成していなかったトップレベルクロックベース記述を自動合成することで、クロックベースシミュレーションを行なうための環境構築期間の短縮化を図ることができる。

【0059】

【発明の効果】

本発明によれば、クロックベースシミュレーション用の記述を自動的に生成することができるシステムレベル設計方法及び装置を提供することができる。

また、アルゴリズムレベル記述をコデザイン用のアルゴリズムレベル記述に変更する際に、回路規模を縮小することができる。その理由は、アルゴリズム記述中における冗長なグローバル変数表現をポート表現に変換しているからである。

【図面の簡単な説明】

【図1】

本発明にかかるシステムレベル設計方法のフローを示す図である。

【図2】

システムレベルアルゴリズムからコデザイン用システムレベルアルゴリズムへの変換を説明するための図である。

【図 3】

グローバル変数の変換を説明するための図である。

【図 4】

グローバル変数の変換とトップレベルクロックベース記述を説明するための図である。

【図 5】

コデザイン装置中のデータを示す図である。

【図 6】

グローバル変数の変換アルゴリズムを示す図である。

【図 7】

トップレベルクロックベース記述の一例を示す図である。

【図 8】

トップレベルクロックベース記述の一例を示す図である。

【図 9】

トップレベルクロックベース記述の一例を示す図である。

【図 1 0】

トップレベルクロックベース記述を説明するための図である。

【図 1 1】

トップレベルクロックベース記述を説明するための図である。

【符号の説明】

- 3 コデザイン用アルゴリズム変換装置
- 5 コデザイン装置
- 9 トップレベル記述生成装置
- 1 2 クロックベース記述生成装置
- 1 4 HW部生成装置
- 2 1 SW部生成装置
- 2 5 クロックベースシミュレーション装置

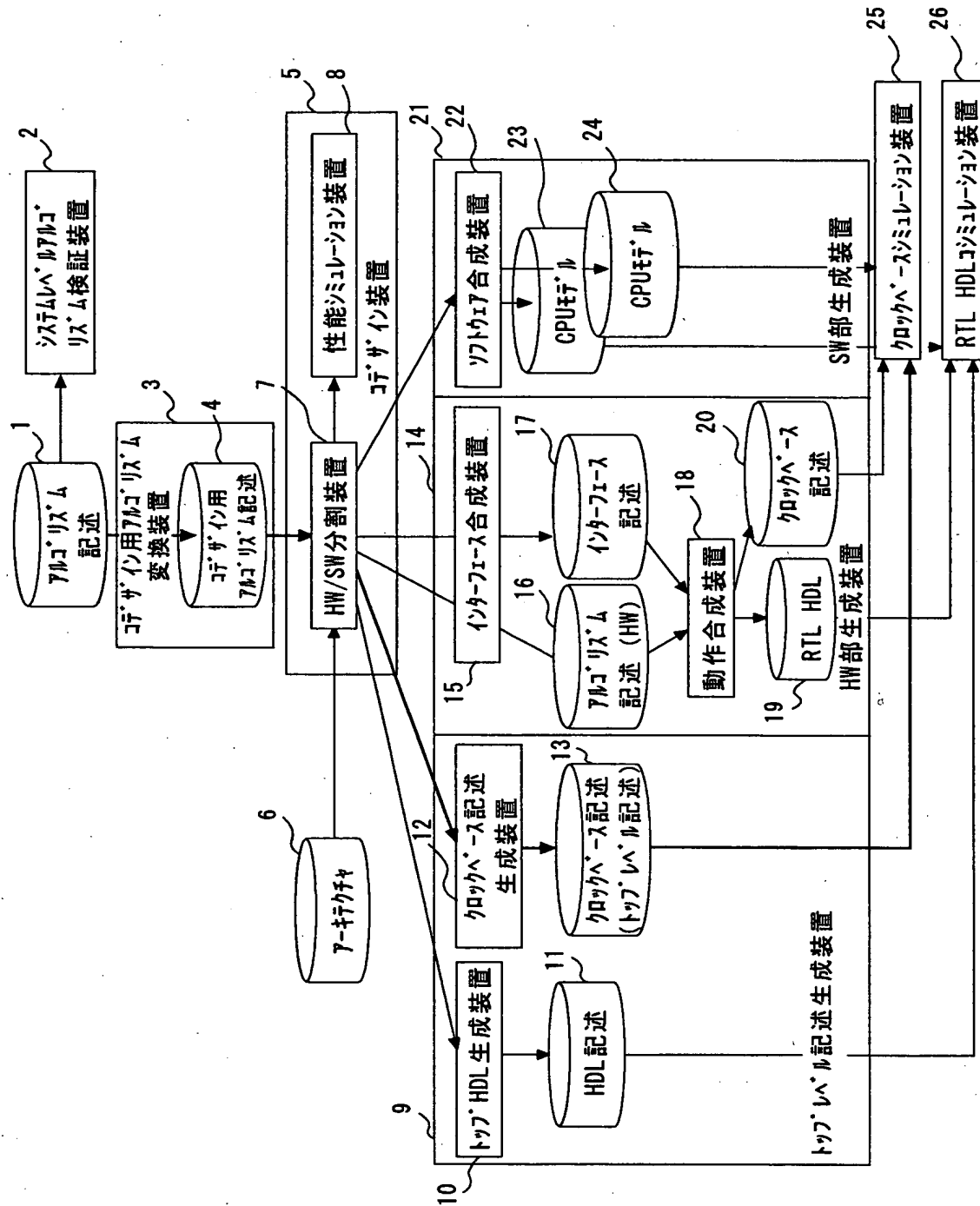
特 2002-213895

26 RTL HDLコシミュレーション装置

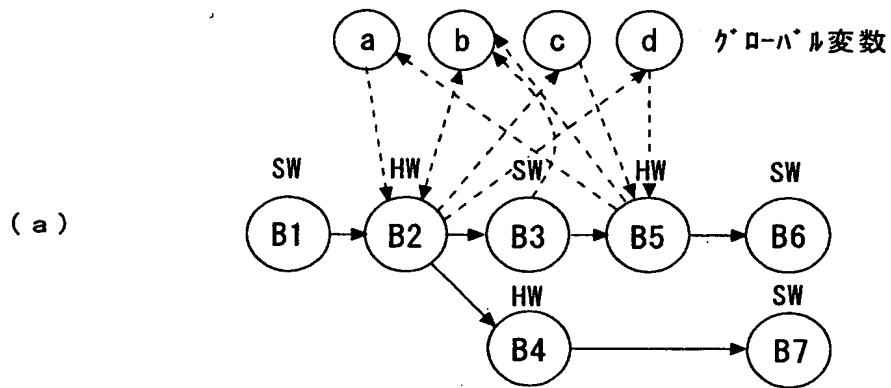
【書類名】

図面

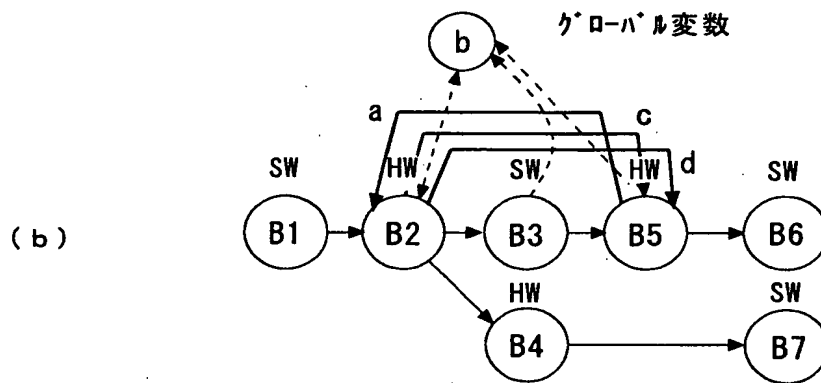
【図 1】



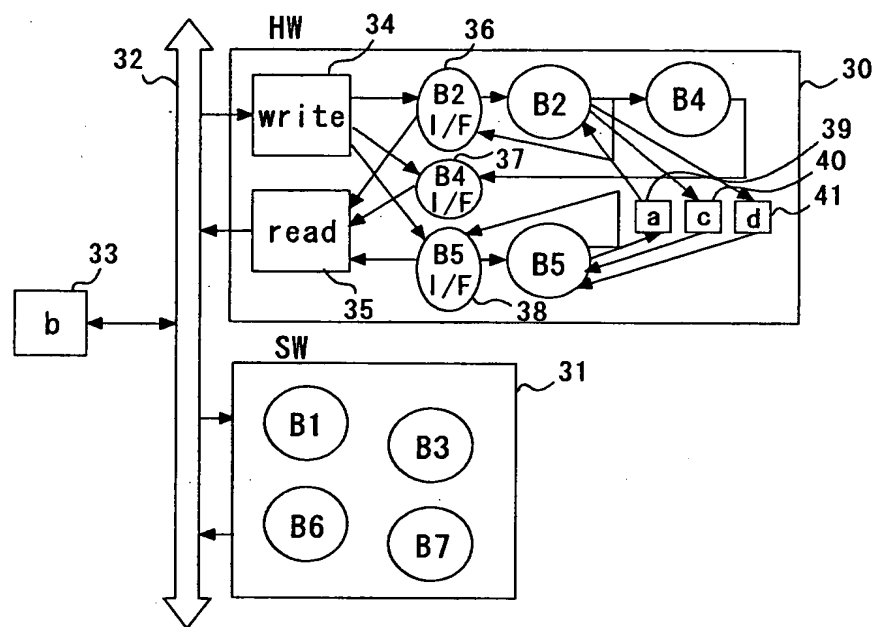
【図 2】



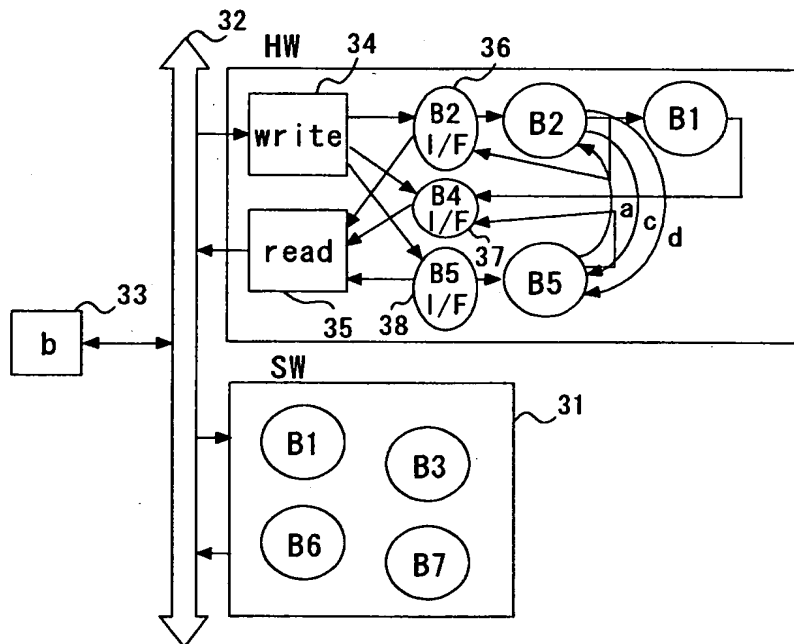
グローバル変数の解消



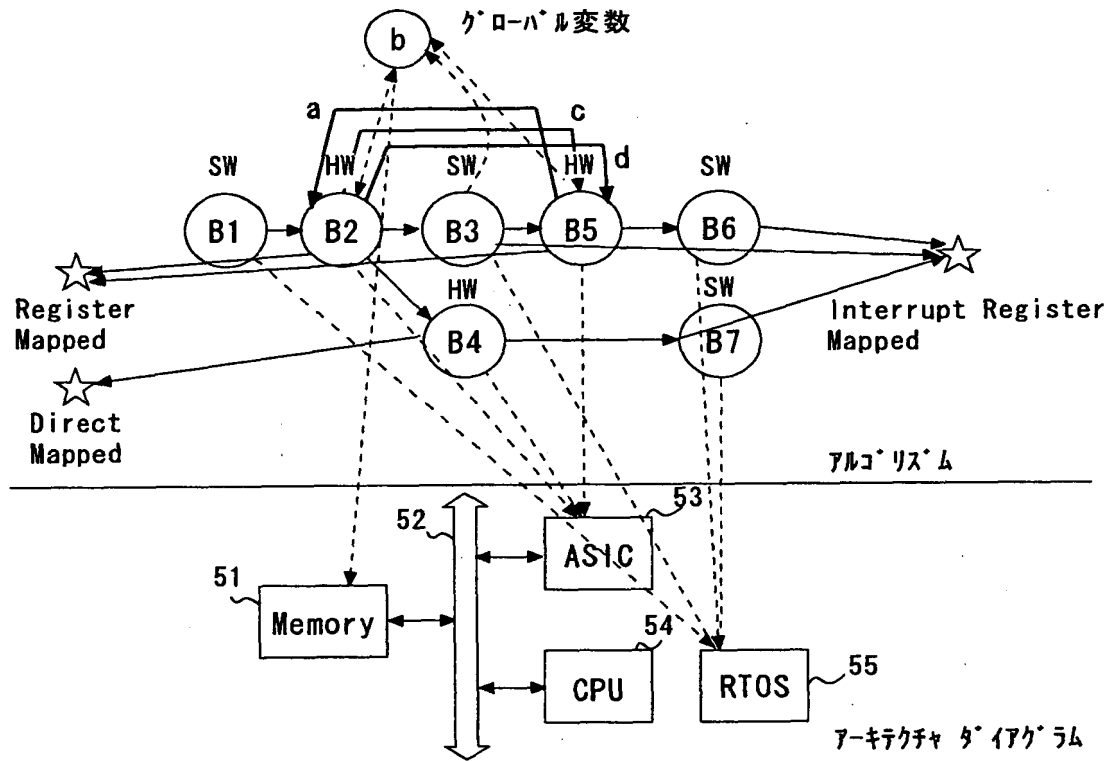
【图 3】



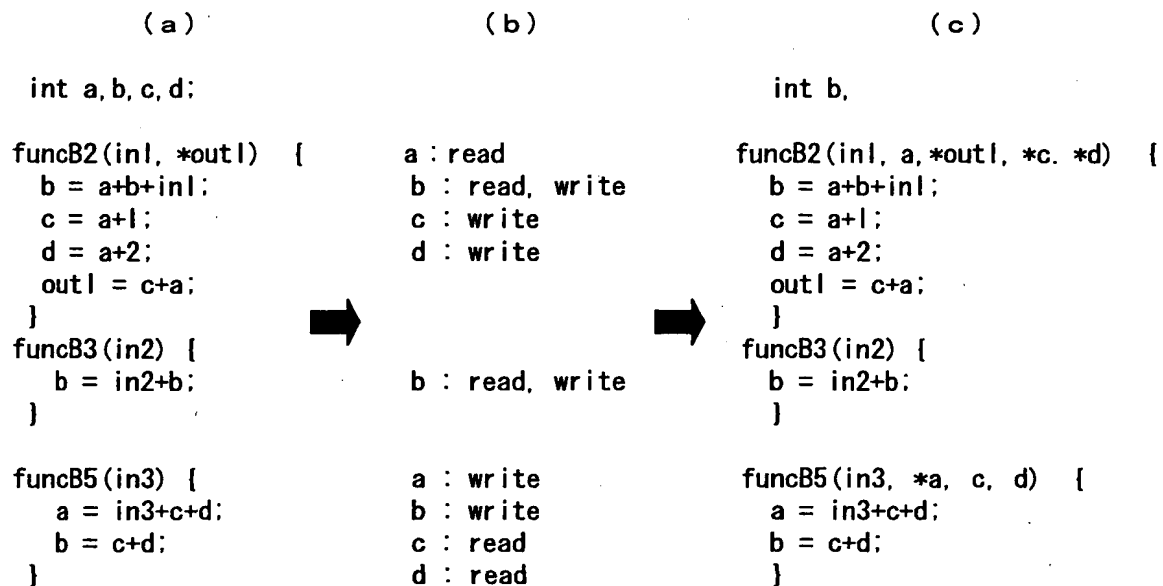
【図 4】



【図 5】



【図 6】



〈合計〉

```

a : read1 回, write1 回
b : read2 回, write3 回 ⇒ b だけ 1 回
c : read1 回, write1 回
d : read1 回, write1 回
    
```

【図 7】

```

BUS_bus 1( "Bus1", arbitor_1, busread_1, buswrite_1)
Int arbitor_1(void);
RDATA busread_1(int add, int byte_count);
Int buswrite_1(int add, int data, int byte_count);

CPU ( "cpu_1", &cpu_1_busif, 0, 0X10000, 0X00ffe000, 0X00000X6000);
BusIntf cpu_1_busif( "cpu 1 Bus I/F", &bus_1, 1);

SDRAM ExtSDRam1( "External SDRam 1", EXTSDRAM1_START_ADD, EXTSDRAM1_SIZE_);

c_cpuifB2 cpuifB2(); c_cpuifB4 cpuifB4(); c_cpuifB5 cpuifB5();
c_B2 funcB2(); c_B4 funcB4(); c_B5 funcB5();

SystemoneScep()
{
    funcB2.OneStep(); cpuifB2.OneStep();
    funcB4.OneStep(); cpuifB4.OneStep();
    funcB5.OneStep(); cpuifB5.OneStep();
}
SystemInit()
{
    funcB2.Init(); cpuifB2.Init();
    funcB4.Init(); cpuifB4.Init();
    funcB5.Init(); cpuifB5.Init();
}
SystemReset()
{
    funcB2.Reset(); cpuifB2.Reset();
    funcB4.Reset(); cpuifB4.Reset();
    funcB5.Reset(); cpuifB5.Reset();
}

```

【図 8】

```

RDATA busread_1 (int add, int byte_count) {
    if ((add < start_bus_address) || (add > end_bus_address)) {
        return(NULL);
    }
    //RAM
    if ((add >= ExternalRAM.StartAddress) && (add < ExternalRAM.EndAddress)) {
        return ExternalRAM.Read(add - ExternalRAM.StartAddress, byte_count);
    }

    //ASIC
    if ((start_sub__address+B2_start_offset <= add)
        && (add < start_sub__address+B2_end_offset)) {
        cpuifB2.cpuread = 0;
        cpuifB2.bus_addin = add;
        rd.Status = cpuifB2.ReadI0Reg();
        rd.data = cpuifB2.dataout;
    }
    if ((start_sub__address+B4_start_offset <= add)
        && (add < start_sub__address+B4_end_offset)) {
        cpuifB4.cpuread = 0;
        cpuifB4.bus_addin = add;
        rd.Status = cpuifB4.ReadI0Reg();
        rd.data = cpuifB4.dataout;
    }
    if ((start_sub__address+B5_start_offset <= add)
        && (add < start_sub__address+B5_end_offset)) {
        cpuifB5.cpuread = 0;
        cpuifB5.bus_addin = add;
        rd.Status = cpuifB5.ReadI0Reg();
        rd.data = cpuifB5.dataout;
    }

    .....

    return (rd);
}

```

【図 9】

```

RDATA buswrite_1 (int add, int data, int byte_count) {
    if ((add < start_bus_address) || (add > end_bus_address)) {
        return(1);
    }
    //RAM
    if ((add >= ExternalRAM.StartAddress) && (add < ExternalRAM.EndAddress)) {
        return ExternalRAM.Write(add - ExternalRAM.StartAddress, byte_count);
    }

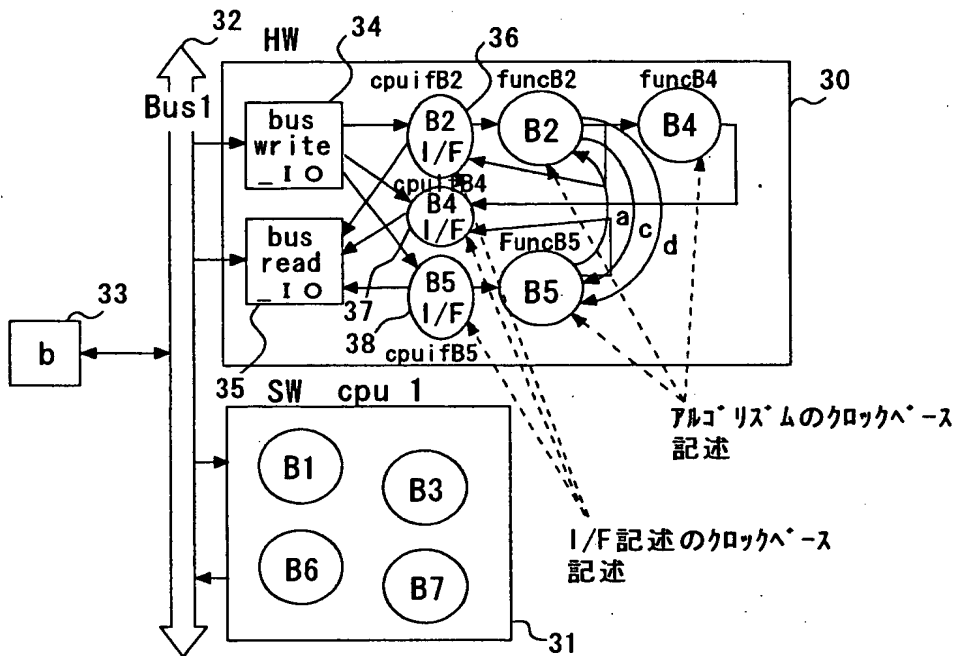
    //ASIC
    if ((start_sub_address+B2_start_offset <= add)
        && (add < start_sub_address+B2_end_offset)) {
        cpuifB2.cpuwrite = 1;
        cpuifB2.bus_addin = add;
        cpuifB2.datain = data;
        rd.Status = cpuifB2.Writel0Reg();
    }
    if ((start_sub_address+B4_start_offset <= add)
        && (add < start_sub_address+B4_end_offset)) {
        cpuifB4.cpuwrite = 1;
        cpuifB4.bus_addin = add;
        cpuifB4.datain = data;
        rd.Status = cpuifB4.writel0Reg();
    }
    if ((start_sub_address+B5_start_offset <= add)
        && (add < start_sub_address+B5_end_offset)) {
        cpuifB5.cpuwrite = 1;
        cpuifB5.bus_addin = add;
        cpuifB5.datain = data;
        rd.Status = cpuifB5.writel0Reg();
    }

    .....

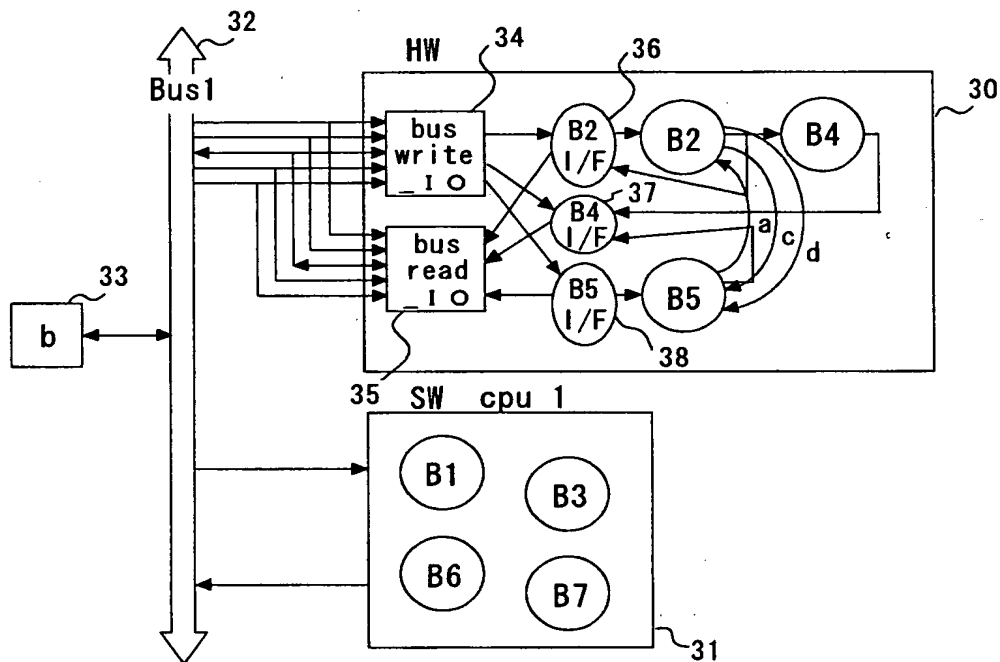
    return (0);
}

```


【図10】



【図11】



【書類名】 要約書

【要約】

【課題】

クロックベースシミュレーション用の記述を自動的に生成すること。

【解決手段】

本発明にかかるシステムレベル設計方法は、コデザイン装置 5 によってシステムのハードウェア部分とソフトウェア部分を分割するとともに、アーキテクチャ情報、マッピング情報及びアドレス情報を含むデータベースを自動生成する。その後、トップレベル記述生成装置 9 にこのデータベースよりアーキテクチャ情報、マッピング情報及びアドレス情報を含む情報を入力する。さらに、トップレベル記述生成装置 9 は、入力された情報に基づいてクロックベースシミュレーション用の記述を自動生成する。

【選択図】 図 1

認定・付加情報

特許出願の番号	特願 2002-213895
受付番号	50201081174
書類名	特許願
担当官	第七担当上席 0096
作成日	平成14年 7月24日

<認定情報・付加情報>

【提出日】	平成14年 7月23日
-------	-------------

【書類名】 出願人名義変更届（一般承継）
【提出日】 平成15年 1月14日
【あて先】 特許庁長官殿
【事件の表示】
 【出願番号】 特願2002-213895
【承継人】
 【識別番号】 302062931
 【氏名又は名称】 N E C エレクトロニクス株式会社
【承継人代理人】
 【識別番号】 100103894
 【弁理士】
 【氏名又は名称】 家入 健
【提出物件の目録】
 【物件名】 承継人であることを証明する登記簿謄本 1
 【援用の表示】 特願 2 0 0 2 - 3 1 8 4 8 8
 【物件名】 承継人であることを証明する承継証明書 1
 【援用の表示】 特願 2 0 0 2 - 3 1 8 4 8 8
 【包括委任状番号】 0218232
【プルーフの要否】 要

出 願 人 履 歴 情 報

識別番号 [000004237]

1. 変更年月日 1990年 8月29日
[変更理由] 新規登録
住 所 東京都港区芝五丁目7番1号
氏 名 日本電気株式会社